

University of Montana

## ScholarWorks at University of Montana

---

Graduate Student Theses, Dissertations, &  
Professional Papers

Graduate School

---

1995

### Heterogeneous internetworking model with enhanced routing security and management functions

Yan Zhu

*The University of Montana*

Follow this and additional works at: <https://scholarworks.umt.edu/etd>

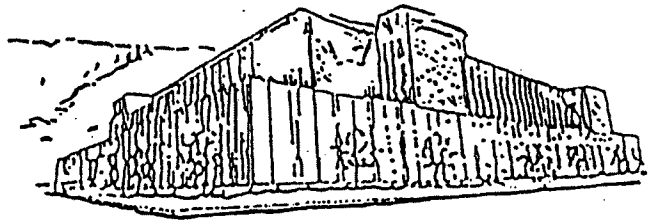
**Let us know how access to this document benefits you.**

---

#### Recommended Citation

Zhu, Yan, "Heterogeneous internetworking model with enhanced routing security and management functions" (1995). *Graduate Student Theses, Dissertations, & Professional Papers*. 5503.  
<https://scholarworks.umt.edu/etd/5503>

This Thesis is brought to you for free and open access by the Graduate School at ScholarWorks at University of Montana. It has been accepted for inclusion in Graduate Student Theses, Dissertations, & Professional Papers by an authorized administrator of ScholarWorks at University of Montana. For more information, please contact [scholarworks@mso.umt.edu](mailto:scholarworks@mso.umt.edu).



Maureen and Mike  
MANSFIELD LIBRARY

The University of MONTANA

---

Permission is granted by the author to reproduce this material in its entirety,  
provided that this material is used for scholarly purposes and is properly cited in  
published works and reports.

*\*\* Please check "Yes" or "No" and provide signature \*\**

Yes, I grant permission

No, I do not grant permission

Author's Signature Yan Zhu

Date July 5, 1995

Any copying for commercial purposes or financial gain may be undertaken only with  
the author's explicit consent.



# A Heterogeneous Internetworking Model with Enhanced Routing, Security and Management Functions

Yan Zhu

B.S., Peking University, P. R. China, 1986

M.S., Institute of Software , Academia Sinica, P. R. China, 1989

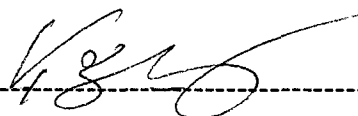
Presented in partial fulfillment of the requirements

for the degree of Master of Science

University of Montana

1995

Approved By



Chairman, Board of Examiners



Dean, Graduate School

JULY 11, 1995

Date

UMI Number: EP40967

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI EP40967

Published by ProQuest LLC (2014). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against unauthorized copying under Title 17, United States Code



ProQuest LLC.  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106 - 1346

Zhu, Yan, M.S., July 1995

Computer Science

A Heterogeneous Internetworking Model with Enhanced Routing, Security and management Functions (83 pp.)

Director: Youlu Zheng 

The world of computing is undergoing changes. Network computing, using PCs, workstations, LANs and WANs, is challenging traditional centralized data processing architectures.

This project aims at building a prototype of a heterogeneous network environment. The problems to be studied in the project are:

- Design the heterogeneous network structure and topology;
- Install and run different protocols and network software;
- Offer automatic routing services among different networks;
- Experiment with all the standard Internet network services, such as ftp, telnet (rlogin), SMTP and email (smail and sendmail) as well as Simple Network Management Protocol in the heterogeneou internetworking environment in addition to NetWare services;
- Design and implement a firewall as the major security mechanism in the heterogeneous internetworking environment.

Three subnetworks running Novell's NetWare, SunOs, BSD386 and Linux operating system are respectively interconnected together with a gateway and a router. A set of internetworking services such as FTP, TELNET, RLOGIN, SMTP and SNMP is offered in the environment.

A router-based filter with packet filtering mechanism is implemented as the kernel of the firewall.

## ACKNOWLEDGEMENTS

I would particularly like to thank Dr. Youlu Zheng for his effort and support, and the initial idea for this project. His expertise was the technical backbone of this paper. I am proud to be his student.

I also would like to thank Professor Jerry Esmay and Professor Dick Walton for their invaluable guidance, and to Dr. Alden Wright for his help with solving the BSD problem.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Overview . . . . .	3
1.2	The Reasons to Build a LAN . . . . .	4
1.3	Problem Statement . . . . .	5
<b>2</b>	<b>A Prototype Heterogeneous Network</b>	<b>6</b>
2.1	Design . . . . .	6
2.1.1	Network Topologies . . . . .	6
2.1.2	Network Protocols . . . . .	8
2.1.3	Network and Network Operating System . . . . .	11
2.2	Implementation . . . . .	13
2.2.1	Physical Connections . . . . .	13
2.2.2	Hardware Architecture and Network Configuration . . . . .	14
2.2.3	Software Components and Configuration . . . . .	16
2.2.4	PC based Software Routing and Gateway . . . . .	20
2.3	Network Services and Network Management . . . . .	22
<b>3</b>	<b>Network Security</b>	<b>24</b>
3.1	General Approach . . . . .	24
3.1.1	Authentication, Authorization and Data Integrity . . . . .	24
3.1.2	Encryption and <i>Kerberos</i> . . . . .	26
3.1.3	Firewalls . . . . .	28
3.2	A Router-based Firewall Using Packet-filtering . . . . .	31
3.2.1	Software Design and Implementation . . . . .	32
<b>4</b>	<b>Summary</b>	<b>46</b>
<b>5</b>	<b>Appendix</b>	<b>46</b>
<b>A</b>	<b>The major steps to set up each machine of the network</b>	<b>46</b>
<b>B</b>	<b>The major customized files on each machine of the network</b>	<b>48</b>
B.1	The customized files on the NetWare Server: . . . . .	48
B.1.1	SYS:/system/autoexec.ncf: . . . . .	48
B.1.2	SYS:/etc/gateways: . . . . .	48
B.2	The customized files on the Linux . . . . .	49
B.2.1	/etc/lilo.conf: . . . . .	49
B.2.2	/etc/rc.d/rc.inet1: . . . . .	50
B.3	The customized files on the 4.3BSD: . . . . .	52
B.3.1	/etc/netstart: . . . . .	52
B.3.2	/etc/sendmail.cf: . . . . .	54
B.4	The customized files on the SunOS: . . . . .	75



B.4.1	/etc/rc.local: . . . . .	75
B.5	The customized file /etc/hosts on each machine: . . . . .	82

# 1 Introduction

## 1.1 Overview

The world of computing is undergoing fundamental changes. Network computing, using PCs, workstations, *LANs* and *WANs*, is challenging traditional centralized data processing architectures. Many computer applications can be successfully moved from minicomputer, mainframe, and even supercomputer platforms onto networked PCs and workstations with significant cost savings and increased performance, flexibility and growth potential. The aggregate power of a *LAN* full of PCs, workstations, and servers allows for the development of new applications and user interfaces that would have been impractical or impossible in the past.

A network is a system that interconnects computers and devices using various media for the efficient use of shared resources and distributed information. It is also a tool for facilitating employee communications and conducting business in a timely fashion through the access and sharing of software and data. A network reduces resource redundancy. This allows every workstation to use the same program from wherever it is located on the network without installing the program on each workstation's disk. With networks, the savings are substantial not only on software, but on printers, disk storage devices and other expensive hardware devices as well.

A network also brings out new problems while it gives people great convenience and savings. Much information, for example personnel files, should not be accessible

to the general public. Account information in a bank is not expected to be shared by every user in the network either. In many situations, as networks grow and are interconnected, the data in one network should not be accessed by the users in another network. Thus, one aspect of management of a computer network, which is essential in order to run the network most efficiently, is how to control and secure the network. The rapid growth of the *Internet* for commercial use makes the problem more serious and urgent.

## 1.2 The Reasons to Build a LAN

A *Local Area Network (LAN)* is a group of computers and peripheral devices located in a close geographic region and connected by a communication medium. The existence of large numbers of microcomputers has been a major factor in the development of LANs.

A *Wide Area Network (WAN)* was originally implemented to make more cost-effective use of large and expensive mainframe computers. Traditionally, a WAN has at least one or more computers central to the operation of the network. Usually the central computer is a timesharing minicomputer or a large mainframe computer. The development of a WAN focuses on specific machines. The communications in a WAN are usually point-to-point.

In contrast, the development of a LAN focuses on connectivity. It emphasizes the connections of daily used devices such as microcomputers, fax machines, copy machines and printers. It also pays more attention to the interconnections of LANs and global networks. LANs are the currently foundations of computerization for almost every field.

### 1.3 Problem Statement

*The purpose of this project is to establish a prototype of a heterogeneous network environment and assure security on it.*

There is no doubt that connecting computers and other devices together makes the computing environment more powerful, efficient and cost-saving. At the same time, it is also true that a cluster of networked computers is more complicated than individual computers. A network runs more software as the interface between users and networks. There are many more facilities to be managed in a network than in a stand-alone personal computer. It is not as easy for a system administrator to master the software and data processing in a network as in a centralized computer system.

This project aims at building a prototype of a heterogeneous network environment to show:

- What the principles of networking are;
- What basic software networking requires;
- What kinds of services a network provides;
- How different protocols and internetworking software work together;
- How a network can be managed and maintained with security.

Additionally, there are many problems arising with computer internetworking. Network security in an internetworked environment is one of the most important issues. A prototype heterogeneous network is an ideal experimental facility for network security problem investigation.

In summary, the problems to be studied in the project are as follows:

- Design the heterogeneous network structure and topology;

- Install and run different protocols and network software;
- Offer automatic routing services among different networks;
- Experiment with all the standard Internet network services, such as ftp, telnet (rlogin), SMTP and email (smail and sendmail) as well as Simple Network Management Protocol (SNMPv2) in the heterogeneous internetworking environment.
- Design and implement a firewall as the major security mechanism in the heterogeneous internetworking environment.

## **2 A Prototype Heterogeneous Network**

### **2.1 Design**

#### **2.1.1 Network Topologies**

There are a variety of ways in which networks can be organized, and most networks are in a constant state of change and growth. If the computer network has a main-site or host computer that does all data processing for one or more remote computers, it is a centralized network. If there are remote computers processing jobs for end-users, as well as a main-site computer, then that is the beginning of a distributed network. A distributed network can be either centralized or dispersed, but a network that does not involve distributed processing can only be centralized since all data processing is done on a main-site computer.

Several network topologies can be selected as a networking structure, such as point-to-point (centralized), star (centralized), ring (distributed), bus structure (distributed) and hierarchical (distributed).

Point-to-point has only a computer, a communications line (direct or through the telephone system), and one computer at the end of the line. This was the earliest

and classical form of networks, mostly for wide area networks, and is undoubtedly not sophisticated enough to be the network topology for the network environment under consideration.

A star network, or centralized network, is one in which primary computing is accomplished at a single site, with each remote site workstation entering the central system via a single communication line. Although a star network may have other powerful computers at the end of its communication lines, it is not a good structure to show distributed processing on a LAN without an extremely powerful computer.

A ring network is organized by connecting computers in a closed loop. Each node is only linked to those adjacent on the right and the left. Hence, the maximum distance a ring network covers is far larger than that of other topologies such as the bus topology. Advantages of a ring network are that it can be run at high speeds and that the mechanisms for avoiding collisions are simple. But a ring topology does not have the flexibility that bus structures have and often carries a high price tag. Ring networks sometimes use token passing schemes to determine which node may have access to the communications system. *Token Ring (IEEE 803.5)* and *Fiber Distributed Data Interface (FDDI)* which use double ring connection are well known networks using ring topology.

A bus network is configured with branches extending from a central backbone. As a signal traverses the bus — normally a coaxial, fiber optic or twisted pair cable — the connection listens for the signal that carries an address designation. The products of bus systems, such as ETHERNET, are the least expensive and most commonly used. Signal collision and signal attenuation limit the range covered by the bus topology network. The bus structure of ETHERNET is chosen as the LAN topology for this study.

A hierarchical network represents a fully distributed network, in which computers feed into computers, which in turn feed into other computers. The computers used for remote devices may have independent processing capabilities and draw upon the resources at higher or lower levels as information or other resources are required. A hierarchical network is one form of a completely distributed network.

### **2.1.2 Network Protocols**

A protocol is a set of rules that coordinates the exchange of messages between partners. Standards for establishing a network benefit the users of computer network and communications protocols. They force manufacturers to consider quality rather than “gimmicks”.

To establish a network standard, the International Organization for Standardization provides a seven-layer reference model (OSI reference model). At present, this model is generally used as a framework to describe protocol characteristics and functions. The importance of the reference model is that it permits various networks of the same or different types to communicate easily with one another as if they constituted a single network. The layering is based on the principle that any layer may use the services of the layer below it without knowing how the layer below provides these services. The layer above offers specific services of its own. In a layered architecture, each system is viewed as a logical composition of a set of subsystems. Subsystems that are adjacent to one another in the vertical hierarchy communicate through their common boundary. Within each subsystem or layer are entities. When a computer communicates, it passes messages between adjacent layers. Each layer, in this OSI model, defines a layer or level of functions. The lower-level implementation can be hidden to achieve compatibility at some higher levels. It allows the implementation to be focused on its own level functions.

Practically, network architecture varies in several layers in terms of this seven-layer model. Both IPX and TCP/IP are layered protocol suites. Their purpose is to provide a set of communication protocols that are manufacturer-independent. When two computers are going to communicate, they break their messages down into manageably sized packets, then send these packets across the network in a format that other computers can understand and decode.

*IPX*, the Novell NetWare protocol, which stands for *Internetwork Packet Exchange*, supports connectionless datagram delivery. Corresponding to the Network layer of the OSI model, this protocol performs addressing, routing and switching to deliver a packet to its destination. Datagram delivery is not guaranteed, but IPX packets are correctly received about 95 percent of the time. The layer above IPX is responsible for error control and reassembling the packets in correct order.

The motive for developing TCP/IP (Transmit Control Protocol/Internet Protocol), is to broaden the base of hardware and software from which to choose for the purpose of competitive binding, rather than be committed to a proprietary networking system. TCP/IP is a set of protocols unrelated to the physical and data link layers. It can be implemented on top of any physical data communication medium. TCP provides the packet sequential, error control and other services required to provide reliable end-to-end communications. IP takes the packet from TCP and passes it through whatever gateways are needed for delivery to the remote TCP layer through the remote IP layer.

Figure 1 shows the structures of these two protocols comparing the ISO's OSI reference model.



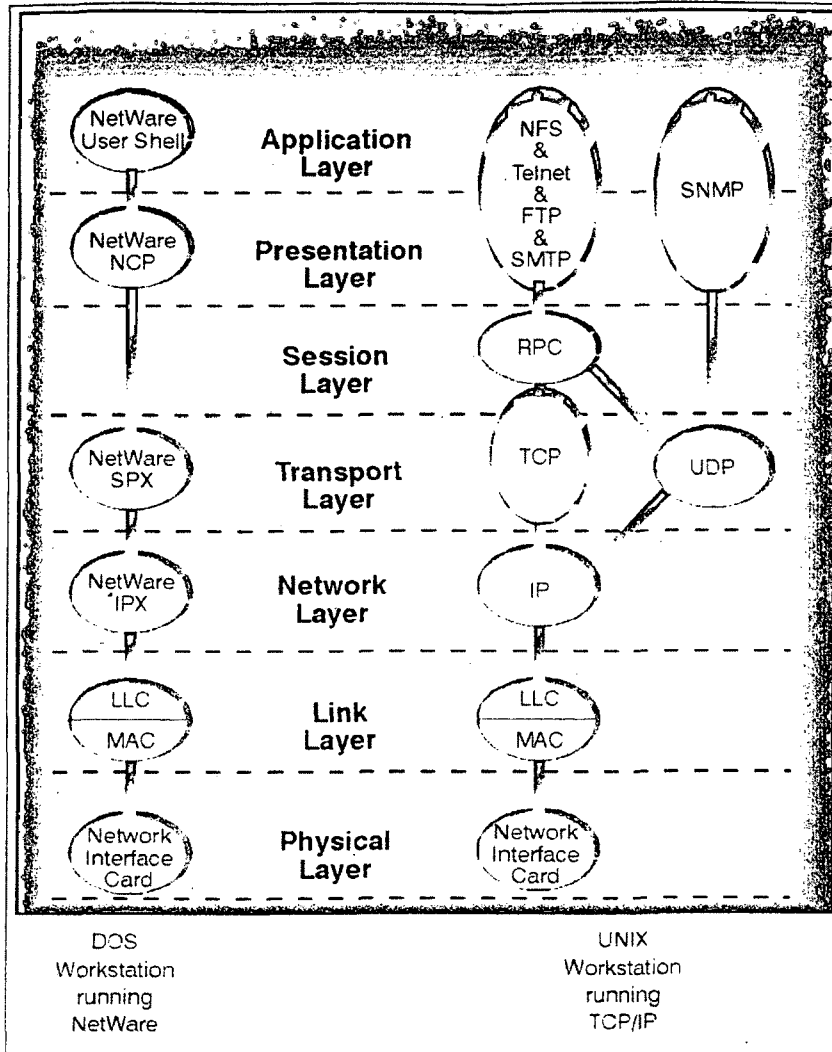


Figure 1: The comparison of three protocols — TCP/IP, IPX and ISO's OSI.

Compared with the OSI seven-layer model, the lowest two layers of IPX and TCP/IP are the same. For the network layer in OSI, NetWare and UNIX TCP/IP are accomplished by IPX and IP, respectively. For the transport layer in the OSI model, NetWare offers SPX and UNIX TCP/IP offers TCP and UDP. NetWare has no functions in the session layer of OSI model, while UNIX TCP/IP has RPC (Remote Procedure Call). For the presentation and application layers in OSI model,

NetWare provides NCP as well as User Shell correspondingly and UNIX TCP/IP provides NFS, Telnet, FTP, SMTP and SNMP across these two layers.

### **2.1.3 Network and Network Operating System**

When designing a LAN, before connecting all computers and related resources together, it must first be determined in which way it is to be used, what kind of protocols to use for transmitting data, what kind of network operating system to use for control of the network, and which networking services to be offered.

There are two major LAN software design configurations. One is peer-to-peer and the other is client-server. In a peer-to-peer network architecture, any computer may make a request of any other computer. There is no dedicated file server, nor any machine dedicated to only one task. Peer-to-peer is usually considered suitable only for wide area networks. The largest network in the world, the Internet, works with the peer-to-peer method for internetworking. In a client-server network architecture, client machines make requests of another machine called the server. Both methods are used in different networks in the prototype networking environment under study.

When looking for a typical client-server model, NetWare is an easy choice. Novell's *NetWare* running IPX protocol is one of the most widely used local area network (LAN) software in the business world. Novell NetWare (but not NetWare Lite) uses the client-server architecture based on PC and UNIX clients requesting services from the NetWare file server and print server. NetWare is mature, robust and priced reasonably for the IBM PC compatible environment. It can also be conveniently connected to other popular internetworking protocols.

On the peer-to-peer side, TCP/IP and related protocols running on numerous remotely located LANs are the base of the information super highway. Millions

of computers and networks are interconnected into the *Internet* running TCP/IP protocols. UNIX is a popular, sophisticated and complete operating system. With the development involving so much time and so many people, it provides many network services. UNIX along with the aforementioned related network products can work in a peer-to-peer architecture.

Another reason to choose these two kinds of network software is that IPX and TCP/IP are the two major and most representative network protocols in the computer and network industry. Also, using these two protocols makes the environment a heterogeneous network.

Using two different protocols creates a crucial problem — how to communicate between the two. Fortunately, NetWare allows operating two protocols on one network interface card. NetWare version 3.1x and 4.x support TCP/IP in four ways:

- A NetWare file server can act like a UNIX server:

Running NetWare *Network File System (NFS)* lets UNIX workstations hook up to a NetWare server and access the files on a server's hard drive.

- A NetWare file server can act as a router for an IP network:

The NetWare server will behave like an IP router if users install the TCPIP.NLM and connect the server to an IP network. The server can route between multiple IP networks or between a NetWare network using the IPX protocol and a UNIX network running TCP/IP using two network interface cards.

- NetWare LAN WorkPlace and LAN WorkGroup allow DOS workstations to talk to a UNIX server:

LAN WorkPlace allows users to send data across an IP network connection to a distant NetWare server.

- NetWare can sneak NetWare packets across an IP network with the IP tunneling feature:

Using the IP tunnel feature, a NetWare server can communicate with a second NetWare server across an IP backbone. The tunnel feature sends the NetWare IPX packets across the IP backbone embedded in IP packets.

With these methods, heterogeneous networks can run both IPX and TCP/IP at the same time simply and smoothly.

## **2.2 Implementation**

### **2.2.1 Physical Connections**

According to the layered network structure, the lowest level is a hardware base for a network, which defines the physical link between computers and networks. In addition to computers which run everyday applications, network interface cards and transmission media — cables are needed for the connection.

A network interface card (NIC) installed in a computer links a computer, whether functioning as a file server, workstation or gateway, to a network or another communication device. A socket at the back of a card provides a connection to any network cable. There are many different kinds of network cards which are usually independent of computer type, network protocols and higher layer network software. Each card has its own driver. Almost every operating system has built-in drivers for most widely used cards to simplify the installation. Generally speaking, there are three basic steps to install a network card: First, insert a card into a computer; Second, load its own driver; Third, activate the driver and modify the operating system to recognize the card. In the experimental network, four types of NICs are used: an S-bus ETHERNET card for a Sun 3-110 workstation, an SMC Elite 16

ETHERNET card (Western Digital 8003 compatible) for a 486 running BSD 386 UNIX, NE2000 network cards for a 486 running Linux and PC 286s, and NE2000 compatible cards for the NetWare Server, an intel 386 computer.

A variety of media are used to provide LAN communication services. The most common are twisted pair copper cables and coaxial cable (for either baseband or broadband). As the engineering problems involved in using fiber optics have been resolved, this technology has been gaining popularity and will probably overtake coaxial cables as the main stream medium. However, there is no single best transmission medium. The selection of a medium is dependent on the use. Twisted pair cable (10baseT) is usually wired in a star pattern with a repeater hub in the center of the star and workstations at the end of the tentacles. Fiber optic cable has been limited to the high-speed and high-security installations because of the high cost of materials and installation. Coaxial cable is mostly used in bus topology networks. Thin coaxial cable (10base2) is cheaper and computers are more easily connected. A computer or other network device can be attached to the bus cable through a T-connector at the back of its NIC. One terminator is attached to each end of the bus. This configuration may become irritating when problems occur because it is very difficult to isolate problems on a bus. Because of the resource availability, coaxial cable is used for the experiment.

### **2.2.2 Hardware Architecture and Network Configuration**

To demonstrate how different kinds of software (such as router, gateway, Simple Network Management Protocol (SNMP), etc.) and different types of protocol work, a limited number of computers are divided into three groups as three LANs. A router and a gateway are used to connect the three LANs together. Figure 2 shows the hardware architecture and network configurations.

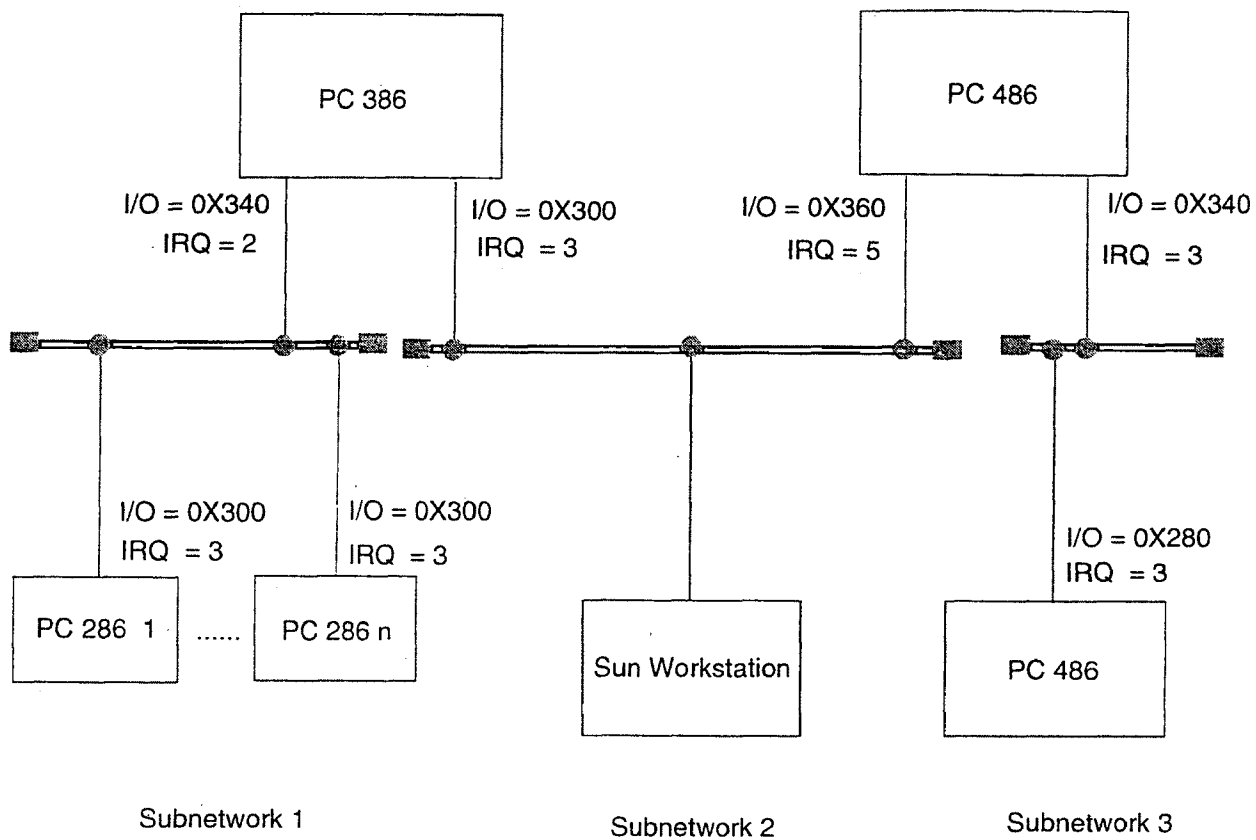


Figure 2: The Hardware Architecture and Network Configuration for Three Internetworked LANs.

The configuration for each node is as follow:

The Sun workstation works as a node. The network card comes with the workstation which uses default I/O base address and default interrupt.

The NetWare Server which is a PC386 works as a gateway between network 1 and network 2. It uses two NE2000 compatible cards. The first is a YCL card with

I/O base address 0X340 and IRQ 2. The second is an AE-200JL-N with I/O base address 0X300 and IRQ 3

A 486 PC works as a router between network 2 and network 3. It has two NE2000 cards attached. The configuration of one card is 0X340 for I/O base address and IRQ 3. The other uses 0X360 for I/O base address and IRQ 5.

The BSD computer is a 486 PC that works as a node. The SMC network card is installed, on which the I/O base address is 0X280 and IRQ is 3

Two 286PCs work as NetWare's clients. They both use NE2000 cards with default configuration, I/O base address 0X300 and IRQ 3.

### **2.2.3 Software Components and Configuration**

The basic networking requirement of the experimental project is that the network software and the internetworking software run seamlessly in the environment to provide enhanced routing, security and management functions in addition to all standard network services. Three UNIX workstations, running BSD386, Linux and SunOS, and a NetWare LAN, including a server and several PCs running MS-DOS and MS-Windows, are organized into three networks as in Figure 3.

Network 1 is a client-server LAN, running NetWare 3.12 with both IPX protocol and TCP/IP protocol. IPX communication protocol is used to connect clients to the NetWare server. The TCP/IP protocol functions on the server to connect it to the other two LANs. Client applications on MSWindows and on MSDOS such as *rconsole*, *login* etc. are installed in the two PC clients.

Both network 2 and network 3 are peer-to-peer networks running UNIX-based software. Network 2 includes a Sun workstation running SunOS, a 486 PC running

Linux and the NetWare server (the gateway to network 1). Network 3 is comprised of a 486 PC running BSD 4.3Net2 and the Linux 486 (the router to network 2). The TCP/IP is chosen as the network protocol on network 2 and network 3. The NetWare server works as a gateway between network 1 and network 2, and the 486 PC with Linux functions as a router between network 2 and network 3.

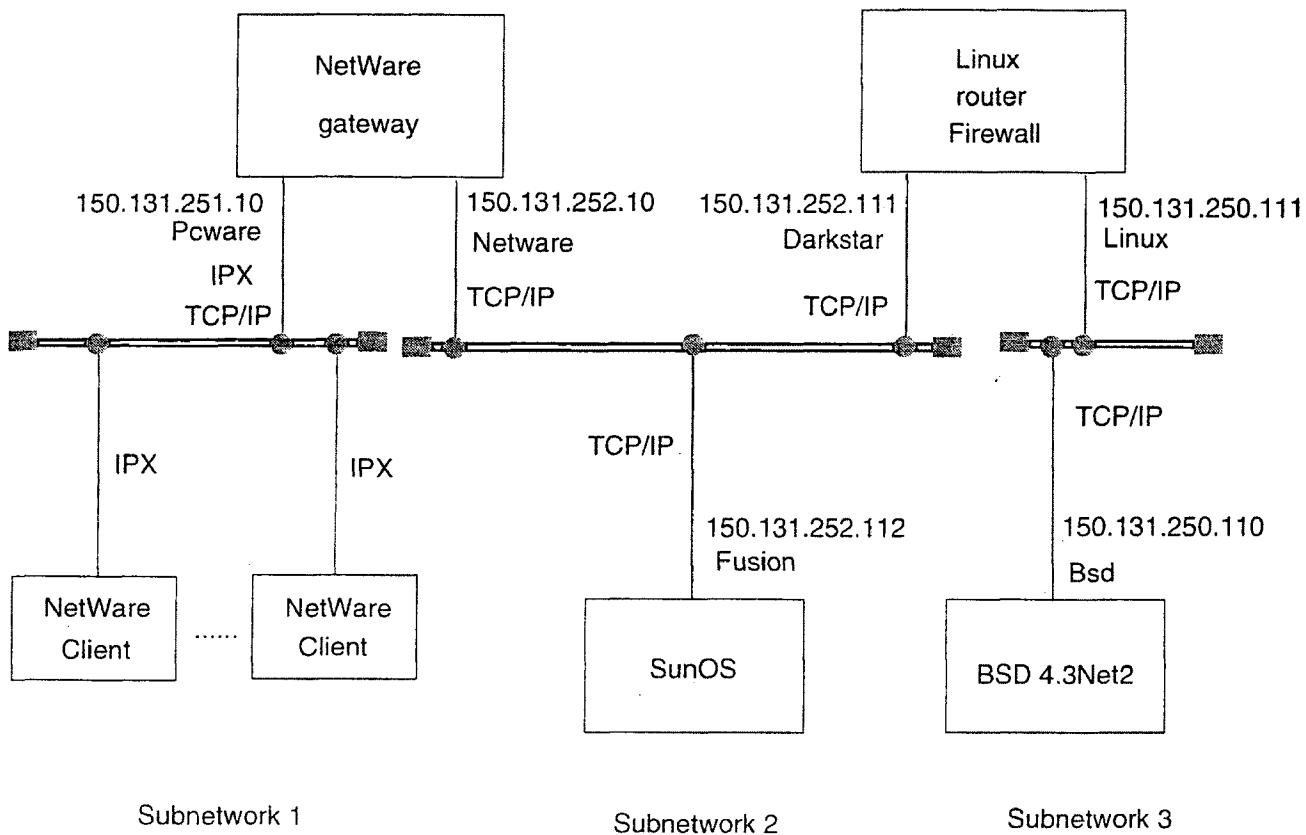


Figure 3: The configuration and connection of a prototype internet with three subnetworks.



To make the internetworked networks communicate smoothly, an IP address should be assigned to each computer and a routing table should be set up on each node. On that basis, higher level protocols and facilities can run.

There are several levels of addressing required for TCP/IP:

- An application port address or port number.
- An Internet address used by IP.
- A hardware network address which uniquely identifies each network interface card and is assigned by the card manufacturer.

A port number is used for addressing at the TCP and UDP (User Datagram Protocol) level (UDP is the connectionless counterpart of TCP), which is a 16-bit field. Different applications have different port numbers. A host computer may theoretically establish up to 65535 different TCP connections concurrently. TCP uses port numbers to identify the ultimate destination within a machine.

UDP also uses port numbers for addressing, but TCP and UDP port numbers are independent. Usually, the designers have chosen to use the same integer port number for any service that is accessible from both UDP and TCP. The interface between TCP and the user accessing an Internet application is through the port address on the host node. The combination of port and global network addresses uniquely identifies a process within an environment of multiple networks and multiple hosts. Some services, such as *FTP* and *rwho*, are assigned reserved or well known port numbers. The port numbers assigned in a UNIX system can be found in the file `/etc/services`.

For internetworking to take place, IP must be able to route according to the global Internet address specific to each node. An Internet address or IP address

is a four-byte address assigned to each network device. IP addresses are globally unique and hardware-independent. They are designed to facilitate the process of routing packets from one network to another so that machines on different physical networks can communicate with each other.

An IP address is four bytes long and is divided into a network portion and a host portion. The network portion identifies a logical network to which the address refers. Routing decisions are made on this information. The host portion identifies a machine on that network.

There are three classes of IP addresses: A, B and C. They differ in the way that bytes are allocated between the host and network portions. If the network portion of an address is denoted by N, and host portion by H, class A addresses interpret the bytes as N.H.H.H, class B as N.N.H.H and class C as N.N.N.H. The class of an address can be determined by examining its first byte. The first byte of class A is between 1 and 126, that of class B is from 128 to 191 and that of class C is from 192 to 223. 127 is for loopback. Class A and B addresses are usually subdivided in a special way: subnetting, in which a part of the host portion of an address is used to extend the network portion. For example, the four bytes of a class B address would normally be interpreted as N.N.H.H. After subnetting is used to assign the third byte to the network number rather than the host number, the address would be interpreted as N.N.N.H. This turns a single class B address into a class C like address. In this way, a mask 255.255.255.0 instead of 255.255.0.0, is used in this project.

In the UNIX system, *ifconfig* is used to assign the IP address, mask and broadcasting address to an interface card. In NetWare, command *bind* is used to assign the IP address. Since IP addresses are long, seemingly random numbers, they may be difficult to remember. Almost every system allows text names to be associated

with an IP address, which are defined in the file `/etc/hosts`.

#### **2.2.4 PC based Software Routing and Gateway**

In many organizations, LANs cannot operate independently. They must be linked either to other LANs or to a WAN. Different kinds of LANs have different implications for interoperability. LANs are particularly important in that it is a LAN that many workstations are connected to as the first stage in a distributed networking and computing environment. However, LANs can rarely exist independently of large networks, particularly in large organizations. The appropriate solution to connect LANs is to use a gateway or router. 'Gateway' has become the term used to designate the hardware and software necessary to make two technologically different networks communicating with one another. Thus, a gateway needs to execute a conversion of protocols. The term router is often used to denote the hardware and software necessary for two physical networks using the same or similar technology to communicate.

Both gateways and routers provide all interconnections among physical networks and switch packets between different sections of the network. They operate at the IP level and are capable of making routing decisions based on information in routing tables. There are dedicated hardware routers with very high performance as well as an extremely high price. However, a host computer with appropriate software can also accomplish routing .

In the software based router, routing is the process of directing a packet through the path of networks that provide the link between its source and destination. Routing information is stored in a table in the kernel. Each table entry has several parameters, including a reliability field that determines routes when the table contains

conflicting information. To route a packet to a particular address, the kernel picks the most specific of the available rules. If there is no relevant route and no default, a “network unreachable” error is returned to the sender.

Packets come in on one interface card and are either delivered locally or compared to the routing table to determine where they should be forwarded. The difficult part is to make sure that the routing table contains the right data. The IP routing, which is determined by searching the routing table and deciding which interface to send a packet out, is a *routing mechanism*. This differs from a *routing policy*, which is a set of rules that decides which routes go into the routing table.

The maintenance of routing tables can be done statically, dynamically or with a combination of the two approaches. A static route is one that can be entered explicitly using the command *route*. Static routes should stay in the routing table forever. This method is very inflexible in respect to changes in the network configuration. Dynamic routing is performed once every 30 seconds by a daemon process that maintains and modifies the routing table. Routing daemons on different hosts communicate to discover the topology of the network and to figure out how to reach distant destinations. In the UNIX environment, the processes that maintain and update routing information are two daemon processes, *routed* and *gated*, based on the well known *Routing Information Protocol (RIP)* and *Open Shortest Path First Protocol (OSPF)*. In addition, ICMP redirection can also add a route entry into the routing table.

In this network prototype, *routed* runs on the SunOS, BSD and the NetWare server (*routed* parameter is set in the “on” position) to dynamically manage the routing table as well as establishing static entries when machines are started. In the Linux computer, a static routing table is used and managed manually.

## 2.3 Network Services and Network Management

Much of the impetus for the development of TCP/IP is the need for internetworking services — the ability of an end-user to communicate through a local machine to some remote machine or remote end-users. The traditional TCP/IP services are supported by the appropriate protocols, such as:

- The *File Transfer Protocol (FTP)*, which allows the transfer of files from one computer on the Internet to another computer on the Internet.
- The *Network Terminal Protocol (TELNET)*, which provides a means for allowing a user on the Internet to log onto any other computer on the network.
- *Simple Mail Transfer Protocol (SMTP)*, which allows users to send messages to one another on the Internet.
- In a TCP/IP Internet, IP routers form the active switches that managers need to examine and control. The *Management Information Base (MIB)* based *SNMP* and *SNMPv2* are the prevalent network management protocols.

Each of the services implied by these protocols is offered by this internetworking prototype except SNMPv2

To use FTP, TCP port 21 is fixed as the command channel and TCP port 20 as the data channel. In the UNIX system, the protocol consists of *ftp* and the server daemon process *ftpd*.

FTP differs from other file transfer programs in many respects. The most prominent differences include the use of separate channels for control information and data and the fact that FTP data transfers do not run in the background (work without a spooler). Now FTP is available for each node running UNIX software to transfer

files between each other.

TELNET is intended to provide access, in the form of a terminal session, to a computer connected to the network. The TELNET service is attached to TCP port 23. UNIX processors currently incorporate the command *rlogin*, which offers almost the same functionality as TELNET but provides better support for the UNIX environment. Both TELNET and RLOGIN run on the prototype networks smoothly.

FTP and TELNET are configured in the file */etc/services* and */etc/passwd*.

TCP port 25 is defined for SMTP. Like FTP and TELNET, SMTP is desirable for its simplicity. It incorporates many features of FTP.

In the UNIX system, SMTP is implemented by the program *sendmail*. This is a very complex program which can communicate with mail services other than SMTP and which, to some extent, also operates as a gateway between different mail systems. It is possible for *sendmail* to function only as an SMTP server, but also as an SMTP client. However, users never use *sendmail* directly, but usually *pine* or *mail*, which controls and simplifies the processing of a message. *Sendmail* is only activated to forward the message. If forwarding is not immediately possible, the message is entered into an output queue. Regular attempts are then made to forward the messages from there to the destination.

A configuration file, *sendmail.cf*, is used to control *sendmail*, which makes the program very adaptable. In addition to the definition of the local-mail-forwarding program and many other uses, this file also contains the commands for converting the address for the connected mail system. File *aliases*, which may be used to create distribution lists and for forwarding requests, is edited and then converted into an indexed database using the command *newaliases*. Alias names may also be used to send a message to a program, for example, to set up an automatic answering service.

The prototype networks provide complete email services.

As the size of networks increases, network management becomes an increasing challenge. The task of network management is to monitor the elements of the network in order to identify malfunctions, faults or structural changes at an early stage and to take countermeasures. In some network architectures, this also includes the control of access to the network and the allocation of information for metered accounting. The basis for the Internet activities in this area is *Simple Network Management Protocol (SNMP)*, which is based on the manager/agent paradigm. SNMP is used by a manager which accesses data in the management information base (MIB) implemented in the agent.

SNMP on NetWare is automatically loaded when TCP.NLM is started.

## **3 Network Security**

### **3.1 General Approach**

Almost all network software provide ways of maintaining network security and system integrity. Network security falls into three broad sets. The first set focuses on the problems of authorization, authentication and integrity. The second set focuses on the problems of privacy, and the third set focuses on the problems of availability by controlling access, i.e. to guarantee that outsiders cannot prevent legitimate data access by saturating a network with traffic.

#### **3.1.1 Authentication, Authorization and Data Integrity**

Authentication is the process of verifying identification. Many servers are configured to reject a request unless the request originates from an authorized client. When

a client first makes contact, the server must verify that the client is authorized before granting services. Authorization involves determining whether a user has legitimate access to a resource and allowing the owner of a resource to define who has what type of access to the resource. In an unsecured network environment, such as the Internet, without reliable authentication and authorization mechanisms, many services are unavailable.

There are a variety of authorization schemes. Access control is widely used. Access control lets network managers restrict the use of specific files, directories, and printers. All access control software require users to identify themselves with a password before they can launch applications. The most popular package offers audit trails to monitor user activity including which systems and files they have used, automatically logs users off the network, and prevents important system files (such as CONFIG.SYS and AUTOEXEC.BAT) from being altered. That feature is particularly valuable because those files contain the commands to run the security package. Some access control packages lock up the PC if they detect that systems files have been modified.

Most access control packages generate and store audit trails, which give network managers a way to monitor system activities. Some products only indicate when a user logged on and off. Other packages also indicate which files were opened and closed.

Knowing that an audit trail exists and that changes can be traced back is often enough to discourage anyone tempted to tamper with data. But it is useless if changes can be made under another user's name, which can be accomplished easily enough if employees walk away from their machines while they are logged on . To guard against this sort of problem, most access control packages clear the screen and lock the keyboard when a PC has remained idle for a predefined period. Once



a screen has been blanked, the user must enter a password to regain control of the keyboard.

At present, authentication provides an easier way to administer passwords and user IDs. Users can be required to change passwords periodically and choose passwords that have a minimal length and some kind of combination of uppercase, lower case letters and digits. Authorization also makes it possible to encrypt passwords before they are sent over a network or stored to disk.

Data integrity ensures that data have not been altered or destroyed in an unauthorized manner.

### **3.1.2 Encryption and *Kerberos***

A primary method used to thwart passive hole is the use of encryption technique to make the information unusable without possession of the decoding key. Encryption is accomplished through the use of either a code or a cipher. With a code, a predefined table substitutes a meaningless word or phrase for each message or part of a message. A cipher, in contrast, uses a computable algorithm that translates a bit stream into an indecipherable cryptogram. Cipher techniques can often be more readily automated, and are therefore used more frequently in computer and network security systems.

With conventional encryption, the original data, or plain text, are converted into an unintelligible ciphertext. The conversion is accomplished using an algorithm and a key composed of a bit string that controls the algorithm. The key must be in the possession of both the sender and receiver, so key management becomes an issue. The algorithm must be sufficiently powerful to preclude decipherment of the message based solely on the ciphertext.

From a network perspective, there are two fundamental types of encryption: link and end-to-end. Link encryption implies that data are encrypted independently on each vulnerable communication link. The task of key management is the control of key selection and key distribution in a cryptographic system. A key is a piece of digital information that interacts with encryption algorithms to control encryption of information. The classical approach to encryption/decryption was to use a symmetric algorithm, in which both the sender and receiver of an encrypted message were required to have the same key. The disadvantage of symmetric algorithms is that for security, the algorithm as well as the key must be kept secret. Moreover, the key must be communicated from one person to another, thus creating a security threat. A major advantage, however, is that some system of authentication can be established so that problems involving bogus information can be reduced.

The sophistication of the algorithms has increased over the years. The newer alternative methodology uses an asymmetric system. *Public key* encryption is the outgrowth of this development. Instead of one key, PKC (Public-Key Cryptograph) has two keys, one public and the other private. Moreover, it is impossible to deduce the private key from the public key. A person with the public key can encrypt a message but not decrypt it — only someone with the private key can decrypt the message. PKC algorithms are complicated protocols, not ideally suited to encrypt long messages. A common implementation is to use PKC to transfer the key for another cryptographic algorithm and then use that algorithm to encrypt and decrypt messages.

The advantage of an asymmetric system is that the second, private key is known only to the receiver who calculates it. In such a system, the encryption/decryption algorithms and the encryption (public) key may be made public. The decipherment key is related to the encipherment key, but is actually a random sample of one of a

large universe of potential key values.

*Kerberos* is an add-on authentication system that delivers a higher level of security than afforded by passwords and access control. In effect, Kerberos is commonly used on the Internet: it has also been adopted by OSF as part of its distributed computing environment.

*Kerberos* — named for the three headed watchdog that guarded the gates of Hades in Greek mythology — relies on three components to watch over the network security: a database, authentication server, and *ticket-granting server* (TGS). All three components sit on a single, physically secured server that is connected to the network.

The Kerberos database contains all network user names, their passwords, the network services to which each user has access, and an encryption key associated with each service. The database is the only place on the network where passwords are stored.

The Kerberos authentication server ensures that the person requesting a network service, such as access to an application on a host, is actually authorized. The ticket-granting server, as its name suggests, issues “tickets” to the user once the authentication server has verified identity. To access an application a user needs both a ticket and a second credential, called an authenticator, which is issued by the client workstation. An authenticator consists of the user name, the IP address of a workstation, and the time that a request originates.

### **3.1.3 Firewalls**

Today’s networked world has grown from the bottom up, with millions of new connections originating from personal computers and small networks. We connect our

organizations' networks to thousands of other computer networks. It is no longer possible to know who or what is on the other end of a network connection unless we take extraordinary measures. With much more at risk, we must protect our data and networks with stronger mechanisms than mere passwords. Mechanisms that control the Internet access handle the problems of screening a particular network or an organization from unwanted communication. Such mechanisms can help prevent outsiders from: obtaining information, changing information or disrupting communication on an organization's internal internet. Unlike authentication and privacy mechanisms, which can be added to an application programs, internet access control usually requires changes to basic components of the internet infrastructure.

A single technique has emerged as the basis for internet access control. The technique places a block known as an internet *firewall* at the entrance to the part of the internet to be protected. A firewall partitions an internet into two regions, referred to informally as the *inside* and *outside*. Actually, a firewall examines all incoming and outgoing messages and blocks all unauthorized communication between computers in the inside and the outside.

The firewall — a gateway or router through which all connections are made — offers another layer of security against intruders getting into the network. Information technology also provides a facility that could monitor and restrict employees exchanging information with outside world. Internet firewalls have their roots in control mechanisms and security measures that have long been standard practice in the mainframe community.

Firewalls can be built in several ways, using a variety of mechanisms. There are three of them in common use:

- Router-based filters, or informally called *packet filters*.

Most routers include a high-speed filtering mechanism. It is the simplest approach to create a firewall using a programmable router. In this way, a system manager can further control packet processing, besides offering normal routing. That is, the manager can specify how the router should dispose of each packet. When a packet first arrives, the router passes the packet through its packet filter before performing any other processing. If the filter rejects the packet, the router drops it immediately. A packet filter forms the basis for building block of a firewall – the kernel of a firewall.

- Host computer gateways, or bastions.

An alternative approach is to use a computer instead of a router, which is called a *bastion host*. This offers many more capabilities, including the ability to log all the activity over the gateway. This bastion host computer is a connection between inside and outside networks. It has two conceptual barriers. The outer barrier blocks all incoming traffic except packets destined for services on the bastion host that the organization chooses to make available externally, and packets destined for clients on the bastion. The inner barrier blocks incoming traffic except packets that originate on the bastion host. Each barrier requires a router that has a packet filter. While router-based firewall monitors data packets at the IP level, hosts exert their control at an application level, where traffic can be examined more thoroughly.

- A separate and isolated network.

This method is similar to the bastion system. Instead of interposing a bastion host computer we create another network, an isolated subnetwork that sits between the external and internal networks. Typically, this network is configured so that both the external and internal networks can access it, but traffic across the isolated network is blocked. One advantage of isolated networks is that they can also simplify the establishment and enforcement of new Internet

addresses, especially for large private networks that may otherwise face the prospect of having to undergo significant reconfiguration.

In this project, packet filter is selected as the network firewall model. It solves the problem in two aspects: to permit a manager to configure separate filter actions for each interface, and to combine packet filtering with existing router.

### **3.2 A Router-based Firewall Using Packet-filtering**

Routers work by controlling traffic at the IP level, selectively passing or blocking data packets based on source/destination address or port information in the packet's header. Many commercial routers offer a mechanism that augments normal routing and permits a manager to further control the packet processing. Informally called a packet filter, the mechanism requires the managers to specify how the router should dispose of each datagram. Some routers permit a manager to configure all interfaces, usually specifying datagrams on the IP address, protocol, source protocol port number, and destination protocol port number.

To be effective, a firewall that uses datagram filtering should restrict access to all IP sources, IP destinations, protocols, and protocol ports except those computers, networks, and services the organization explicitly decides to make available externally. A packet filter that allows a manager to specify which datagrams to admit instead of which datagrams to block can make such restrictions easy to specify.

The problem with the router-based approach stems from the variety of different protocols which are used on the Internet.

### 3.2.1 Software Design and Implementation

The Linux operating system supports TCP/IP communication protocol and also allows a computer running Linux to be configured as a router. It is possible to implement a router-based packet filter on the existing router. To take advantage of the Linux routing software, and to make the implementation more structured and easy to use, the packet filter is separated into two parts: One is the modification of the Linux IP software which manages the packet routing in that some new functions are added to control the packets' forwarding; The other is a utility that allows users to specify their packet routing rules, that is how to process every arriving packet.

1. The Linux packet routing software.

The goal of packet routing is to provide a virtual network that encompasses multiple physical networks and offers a connectionless datagram delivery service. IP routing chooses a path over which a datagram should be sent. Loosely speaking, the routing is divided into two forms: *direct delivery* and *indirect delivery*. Direct delivery, the transmission of a datagram from one machine across a single physical network directly to another, is the basis on which all internet communication rests. Indirect delivery occurs when the destination is not on a directly attached network, forcing the sender to pass the datagram to a router for delivery.

Transmission of an IP datagram between two machines on a single physical network does not involve routers. The sender encapsulates the datagram into a physical frame, binds the destination IP address to a physical hardware address, and sends the resulting frame directly to the destination. Because the internet addresses of all machines on a single network include a common network prefix, and because extracting that prefix can be done in a few machine

instructions, testing whether a machine can be reached directly is extremely efficient.

Indirect delivery is more difficult than direct delivery because the sender must identify a router to which the datagram is to be sent. The router must then forward the datagram on toward the destination network. Routers in a TCP/IP internet form a cooperative, interconnected structure. Datagrams pass from router to router until they reach a router that can delivery the datagram directly.

The usual IP routing algorithm employs a *routing table* on each machine that stores information about possible destinations and how to reach them. Because both hosts and routers route datagrams, both have IP routing tables. Whenever the IP routing software in a host or a router needs to transmit a datagram, it consults the routing table to decide where to send the datagram. After taking into account everything about routing, an IP routing algorithm becomes:

```
RouteDatagram (Datagram, Routing Table)
```

```
  Extract destination IP address , D, from the datagram and compute the network
  prefix, N;
```

```
  If N matches any directly connected network address
```

```
    deliver datagram to destination D over that network,
```

```
    (This involves resolving D to a physical address, encapsulating the datagram,
    and sending the frame.)
```

```
  else if the table contains a host-specific route for D
```

```
    send datagram to next-hop specified in the table
```

```
  else if the table contains a route for network N
```

```
    send datagram to next-hop specified in the table
```



```

else if the table contains a default route
    send datagram to the default router specified in the table
else declare a routing error;

```

The algorithm is split around various routines in Linux. The major functions are as follow:

- *int ip\_build\_header(struct sk\_buff \*skb, unsigned long saddr, unsigned long daddr, struct device \*\*dev, int type, struct options \*opt, int len, int tos, int ttl);*  
This routine builds the appropriate hardware/IP headers for the routine. It assumes that if \*dev != NULL then the protocol knows what it's doing, otherwise it uses the routing/ARP tables to select a device struct.
- *unsigned short ip\_compute\_csum(unsigned char \*buff, int len);*  
This routine does all the checksum computations that do not require anything special (like copying or special headers).
- *static struct sk\_buff \*ip\_defrag(struct iphdr \*iph, struct sk\_buff \*skb, struct device \*dev);*  
Process an incoming IP datagram fragment.
- *void ip\_fragment(struct sock \*sk, struct sk\_buff \*skb, struct device \*dev, int i s\_frag);*  
If the IP datagram is too large to be sent in one piece, break it up into smaller pieces (each of size equal to the MAC header plus IP header plus a block of the data of the original IP data part) that will yet fit in a single device frame, and queue such a frame for sending by calling the ip\_queue\_xmit().
- *static void ip\_forward(struct sk\_buff \*skb, struct device \*dev, int is\_frag);*  
Forward an IP datagram to its next destination.
- *int ip\_rcv(struct sk\_buff \*skb, struct device \*dev, struct packet\_type \*pt);*  
This function receives all incoming IP datagrams.

- *void ip\_queue\_xmit(struct sock \*sk, struct device \*dev, struct sk\_buff \*skb, int free);*  
Queues a packet to be sent, and starts the transmitter if necessary. if free = 1 then free the block after transmit, otherwise don't. If free==2 not only free the block but also don't assign a new ip seq number. This routine also needs to put in the total length, and compute the checksum.
- *int ip\_setsockopt(struct sock \*sk, int level, int optname, char \*optval, int opt len);*  
Socket option code for IP. This is the end of the line after any TCP,UDP etc options on an IP socket.
- *int ip\_getsockopt(struct sock \*sk, int level, int optname, char \*optval, int \*opt len);*  
Get the socket options.

## 2. Modification of the existing Linux routing software. New functions added at IP level:

Several routines are added to the original Linux IP software so that when any packet arrives, it is examined before doing normal routing. In the IP software, there are functions which offer the different utilities to other layers of the leveled TCP/IP protocol suite. They deal with the packet's receiving, sending, forwarding, header-building and sum-checking. They are the only interfaces between the IP layer and any other layer. A new function *ip\_fw\_chk* is added to these functions to do extra checking. That is to decide whether a packet should be discarded or not. To make the *ip\_fw\_chk* work, some other new functions are added, too.

The new functions are:

- *extern inline int port\_match(unsigned short \*portptr, int nports, unsigned short port, int range\_flag);*

Returns 1 if the port is matched by the vector, otherwise return 0.

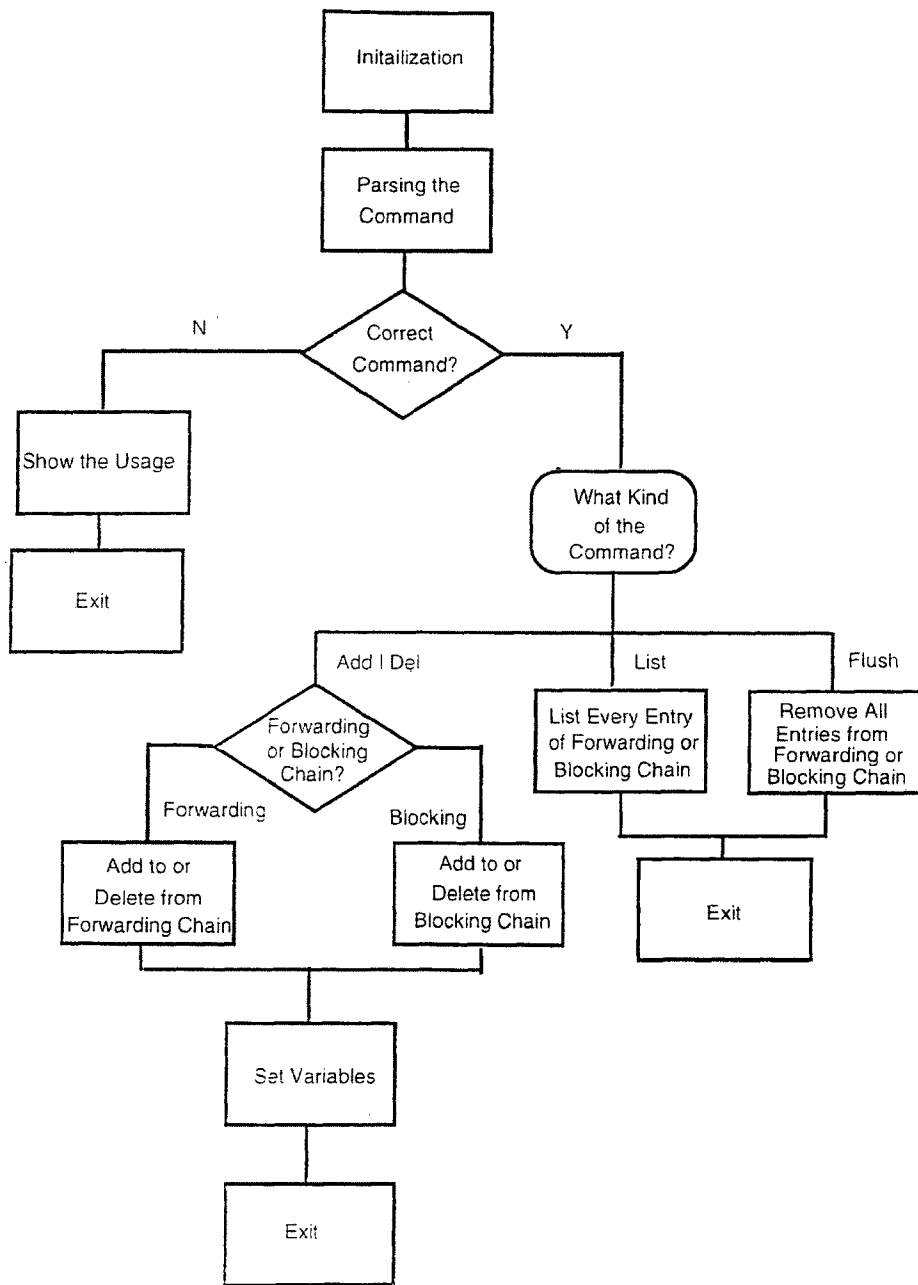
- *int ip\_fw\_chk(struct iphdr \*ip, struct device \*rif, struct ip\_fw \*chain, int policy, int opt);*  
Returns 0 if packet should be dropped, 1 if it should be accepted, and -1 if an ICMP host unreachable packet should be sent.
- *int ip\_fw\_ctl(int stage, void \*m, int len);*  
In this function, *free\_fw\_chain*, *add\_to\_chain* and *del\_from\_chain* are called depending on what the stage is.
- *static void free\_fw\_chain(struct ip\_fw \*volatile \*chainptr);*  
free every element in a chain.
- *static int add\_to\_chain(struct ip\_fw \*volatile \*chainptr, struct ip\_fw \*fowl);*  
add an element in a chain.
- *static int del\_from\_chain(struct ip\_fw \*volatile \*chainptr, struct ip\_fw \*fowl);*  
delete an element from a chain.

### 3. The utility of using the packet-filter software: the user interface ipfw.

Actually, there are three steps to set up a packet filter: first, to decide the security policy; second, express the policy in a format that the computer can understand; third, make the policy work. There is no doubt that it is very important to let the computer know what users want it to do. It is also the most difficult part.

In this practical environment, an independent software package, *ipfw*, is offered for users to specify what they want to do. *Ipfw* maintains two chains which are forwarding and blocking. Through these two chains, a user decides which packet needs to be forwarded or blocked. *Add*, *delete* and *list* are the three types of commands provided. After *ipfw* parses a command, it sets the correspond socket options and variables.

The general algorithm is:



Some important routines of this program are:

- `static char *fmtip(u_long uaddr)`  
change an IP address into the four octets format.

- *static int do\_setsockopt(char \*what, int fd, int proto, int cmd, void \*data, int datalen, int ok\_errno)*  
set the socket options.
- *int get\_protocol(char \*arg, void (\*cmd\_usage) (ipf\_kind), ipf\_kind kind)*  
parsing the command to get the protocol.
- *void get\_ipaddr(char \*arg, struct in\_addr \*addr, struct in\_addr \*mask, void (\*usage) (ipf\_kind), ipf\_kind kind)*  
parsing the command to get the IP address.
- *int get\_ports(char \*\*\*argv\_ptr, u\_short \*ports, int min\_ports, int max\_ports, void (\*usage) (ipf\_kind), ipf\_kind kind, const char \*proto\_name)*  
parsing the command to get the prot number.
- *void add(ipf\_kind kind, int socket\_fd, char \*\*argv)*  
add an entry to the forwarding or blocking chain.
- *void del(ipf\_kind kind, int socket\_fd, char \*\*argv)*  
delete an entry from the forwarding or blocking chain.
- *void list(int socket\_fd, char \*\*argv)*

The usage of this utility is:

- *ipfw <entry-action> <chain entry pattern>*
- *ipfw <chain-action> <chain[s] type>*

These are the *<entry-actions>*:

- *a[dd] b[locking]* - add entry to blocking firewall.
- *d[el] b[locking]* - remove entry from blocking firewall.
- *a[dd] f[orwarding]* - add entry to forwarding firewall.
- *d[el] f[orwarding]* - remove entry from forwarding firewall.

These are the *<chain-actions>*:

- *f[lush]* - remove all entries in forwarding/blocking chains.
- *l[ist]* - show all entries in blocking/forwarding chains.

The *<chain-entry pattern>* is built like this:

- For forwarding/blocking chains:
  - *deny* <proto/addr pattern>
  - *accept* <proto/addr pattern>
- The <proto/addr pattern> options are:
  - all|icmp from <src addr/mask> to <dst addr/mask>
  - tcp|udp from <src addr/mask>[ports] to <dst addr/mask>[ports]
  - <src addr/mask>: <INET IP addr | domain name>[/mask bits | :mask pattern]
  - [ports]: [ port,port...|port:port] where name of service can be used instead of port numeric value.

The <chain[s] type> is:  
forwarding or blocking.

#### 4. An example of using firewall programs

**This file is a script file containing ipfw commands executed on the router Linux:**

Script started on Tue Jun 20 17:36:16 1995

This command shows the ipfw usage

```
# ipfw
```

```
usage: ipfirewall
```

```

l[ist] b[locking]
| l[ist] f[orwarding]
| f[lush] b[locking]
| f[lush] f[orwarding]
| a[dd] b[locking] <type> [iface <addr>] from <src> to <dst>
| a[dd] f[orwarding] <type> [iface <addr>] from <src> to <dst>
| d[el] b[locking] <type> [iface <addr>] from <src> to <dst>

```

```
| d[el] f[orwarding] <type> [iface <addr>] from <src> to <dst>
```

```
#
```

This command lists every entry in the forwarding chain

```
# ipfw l f
```

```
Type      Proto      From      Type      To      Ports
```

```
#
```

This command adds entry which denies all tcp packets from bsd to telnet port of fusion

```
# ipfw add f deny tcp from bsd to fusion telnet
add forwarding deny tcp from bsd to fusion telnet
```

```
#
```

This command disallows any connection from bsd to fusion

```
# ipfw add f deny all from bsd to fusion
add forwarding deny all from bsd to fusion
```

This command lists every entry in the forwarding chain

```
# ipfw l f
```

```
Type      Proto      From      Type      To      Ports
deny      tcp      bsdunix.net3      fusion.net2      any -> telnet
deny      all      bsdunix.net3      fusion.net2
```

```
#
```

This command discards the entry which disallows any connection from bsd to fusion

```
# ipfw del f deny all from bsd to fusion
delete forwarding deny all from bsd to fusion
#
```

This command lists every entry in the forwarding chain

```
# ipfw l f
Type      Proto      From          Type      To          Ports
deny      tcp        bsdunix.net3  fusion.net2  any -> telnet
```

This command flushs every entry in the forwarding chain

```
# ipfw f f
```

This command lists every entry in the forwarding chain

```
# ipfw l f
Type      Proto      From          Type      To          Ports
# exit
# exit
```

Script done on Tue Jun 20 17:44:42 1995

**This file is a script file containing the results with different routing restrictions on BSD:**

Script started on Tue Jun 20 16:20:56 1995

Under the normal condition, any connections such as ping, rlogin and telnet are allowable



```
# ping fusion
PING fusion.net2 (150.131.252.112): 56 data bytes
64 bytes from 150.131.252.112: icmp_seq=0 ttl=254 time=6.719 ms
64 bytes from 150.131.252.112: icmp_seq=1 ttl=254 time=4.265 ms
64 bytes from 150.131.252.112: icmp_seq=2 ttl=254 time=4.300 ms
64 bytes from 150.131.252.112: icmp_seq=3 ttl=254 time=4.273 ms
64 bytes from 150.131.252.112: icmp_seq=4 ttl=254 time=4.246 ms
64 bytes from 150.131.252.112: icmp_seq=4 ttl=254 time=4.246 ms
64 bytes from 150.131.252.112: icmp_seq=5 ttl=254 time=4.258 ms
```

```
--- fusion.net2 ping statistics ---
```

```
6 packets transmitted, 6 packets received, 0% packet loss
round-trip min/avg/max = 4.246/4.670/6.719 ms
```

```
# telnet fusion
```

```
Trying 150.131.252.112...
Connected to fusion.net2.
Escape character is '^]'.
```

```
SunOS UNIX (fusion)
```

```
login: root
```

```
Password:
```

```
Last login: Tue Jun 20 14:18:04 from bsdunix.net3
```

```
SunOS Release 4.1.1 (FUSION_OS) #1: Thu Sep 26 15:14:59 EDT 1991
```

```
/
```

```
FUSION##/(1 )>logout
```

Connection closed by foreign host.

```
# rlogin fusion
```

```
Password:
```

```
Last login: Tue Jun 20 16:27:14 from bsdunix.net3
```

```
SunOS Release 4.1.1 (FUSION_OS) #1: Thu Sep 26 15:14:59 EDT 1991
```

```
/
```

```
FUSION##/(1 )>logout
```

```
rlogin: connection closed.
```

```
#
```

After the command `ipfw add f deny tcp from bsd to fusion telnet` is executed on Linux, the connection from bsd to fusion through telnet is denied. But the other connections such rlogin and ping are still allowable.

```
# ping fusion
```

```
PING fusion.net2 (150.131.252.112): 56 data bytes
```

```
64 bytes from 150.131.252.112: icmp_seq=0 ttl=254 time=5.200 ms
```

```
64 bytes from 150.131.252.112: icmp_seq=1 ttl=254 time=6.145 ms
```

```
64 bytes from 150.131.252.112: icmp_seq=2 ttl=254 time=5.324 ms
```

```
64 bytes from 150.131.252.112: icmp_seq=2 ttl=254 time=5.324 ms
```

```
64 bytes from 150.131.252.112: icmp_seq=3 ttl=254 time=5.304 ms
```

```
64 bytes from 150.131.252.112: icmp_seq=3 ttl=254 time=5.304 ms
```

```
64 bytes from 150.131.252.112: icmp_seq=4 ttl=254 time=5.583 ms
```

```
--- fusion.net2 ping statistics ---
```

```
5 packets transmitted, 5 packets received, 0% packet loss
```

```
round-trip min/avg/max = 5.200/5.500/6.145 ms
```

```
# rlogin fusion
```

Password:

Last login: Tue Jun 20 16:27:30 from bsdunix.net3

SunOS Release 4.1.1 (FUSION\_OS) #1: Thu Sep 26 15:14:59 EDT 1991SunOS Release 4.1

/

FUSION##/(1 )>logout

rlogin: connection closed.

# telnet fusion

Trying 150.131.252.112...

telnet: Unable to connect to remote host: Connection timed out

#

After executing command ipfw add f deny all from bsd to fusion, any connections are disallowable.

# ping fusion

PING fusion.net2 (150.131.252.112): 56 data bytes

--- fusion.net2 ping statistics ---

4 packets transmitted, 0 packets received, 100% packet loss

# rlogin fusion

fusion.net2: Connection timed out

After ipfw f f is executed, that is there is no restriction on routing, any connection from bsd to fusion are allowable.

# ping fusion

PING fusion.net2 (150.131.252.112): 56 data bytes

64 bytes from 150.131.252.112: icmp\_seq=0 ttl=254 time=5.228 ms

64 bytes from 150.131.252.112: icmp\_seq=1 ttl=254 time=4.289 ms

```
--- fusion.net2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 4.289/4.750/5.228 ms
# rlogin fusion
Password:
Last login: Tue Jun 20 16:28:22 from bsdunix.net3
SunOS Release 4.1.1 (FUSION_OS) #1: Thu Sep 26 15:14:59 EDT 1991
/
FUSION##/(1 )>logout
rlogin: connection closed.
# telnet fusion
Trying 150.131.252.112...
Connected to fusion.net2.
Escape character is '^]'.

SunOS UNIX (fusion)

login: root
Password:
Last login: Tue Jun 20 16:33:18 from bsdunix.net3
SunOS Release 4.1.1 (FUSION_OS) #1: Thu Sep 26 15:14:59 EDT 1991
/
FUSION##/(1 )>logout
Connection closed by foreign host.
# exit
```

## 4 Summary

The Internet beckons us in some alluring ways, It promises many advantages in the way of rewards and benefits — connections with a multitude of individuals and organizations and access to information and resources on a scale heretofore unparalleled. And yet hooking up to the Internet can also be the source of significant dangers and risks.

Security is sometimes an elusive goal and can seem unattainable, especially when you think in terms of the exposure that an Internet connection offers. But there are workable, practical solutions today.

## 5 Appendix

### A The major steps to set up each machine of the network

- The major steps to set up a NetWare Server
  1. install the two network cards with the correct configuration;
  2. modify the files:
    - *AUTOEXEC.NCF*: add the two network cards' drivers, assign IP addresses and set the NetWare server's system variables);
    - */etc/hosts*: add every host's IP address;
    - */etc/gateways*: add the router information.
  3. restart the server.
- The major steps to set up a 486 PC running Linux
  1. install the two network cards with the correct configuration;
  2. modify the files:

- */etc/lilo.conf*: set the second network card's driver in Linux;
  - */etc/hosts*: add every host's IP address;
  - */etc/rc.c/rc.inet1*: set the network variables including the routing information and IP addresses of the two network cards;
  - */etc/sendmail.cf*: set the email configuration.
3. run *lilo*, which passes the system variables to the Linux operating system;
  4. reboot.
- The major steps to set up a Sun workstation running SunOS
    1. modify the files:
      - */etc/hosts*: add every host's IP address;
      - */etc/rc.local*: set the network variables including the routing information and IP addresses of the network card;
      - */etc/sendmail.cf*: set the email configuration.
    2. reboot.
  - The major steps to set up a 486 PC running 4.3BSD
    1. install the network card with the correct configuration;
    2. modify the files:
      - */etc/hosts*: add every host's IP address;
      - */etc/netstart*: set the network variables including the routing information and IP addresses of the network card;
      - */etc/sendmail.cf*: set the email configuration.
    3. run *isendmail -bz* to generate the file *sendmail.fc*, another configuration file for email;
    4. reboot.

## **B The major customized files on each machine of the network**

### **B.1 The customized files on the NetWare Server:**

#### **B.1.1 SYS:/system/autoexec.ncf:**

```
file server name SERVER312
ipx internal net 2F28054D
load nmagent
;install the interface for net 150 and 150.131.251.0
load ne2000 port=340 int=2 name=ipxnet
bind ipx to ipxnet net=150
load ne2000 port=340 int=2 frame=ethernet_ii name=ipipxnet
; install the interface for 150.131.252.0
load ne2000 port=300 int=3 frame=ethernet_ii name=tcpxnet
load tcpip forward=yes
bind ip to tcpxnet addr=150.131.252.10 mask=255.255.255.0
bind ip to ipipxnet addr=150.131.251.10 mask=255.255.255.0
;load ipconfig
;load btrieve
;load bspcom
;search add sys:\mhs\exe
;search add sys:\mhs
```

#### **B.1.2 SYS:/etc/gateways:**

```
#
```

```

# SYS:ETC\GATEWAYS
#
#       List of unusual routes which must be added to the routing
#       database statically.
#
# Normally you will not need this file, as most routing information
# should be available through the routing protocols.
#
# Examples.  These entries will not be useful to you.
#net milnet gateway sj-in5                # to milnet through in5.
#net 26 gateway 130.57.6.144 metric 3 active # route to network 26
#net 10 gateway 192.67.172.71 passive      # route to network 10 is passive
#host 130.57.6.40 gateway 192.67.172.55   # route to host 130.57.6.40
#net 150.131.252.0 gateway 150.131.252.10
#net 150.131.251.0 gateway 150.131.251.10
net 150.131.250.0 gateway 150.131.252.111
#

```

## B.2 The customized files on the Linux

### B.2.1 /etc/lilo.conf:

```

# LILO configuration file
# generated by 'liloconfig'
#
# Start LILO global section
boot = /dev/hda

```



```

#compact      # faster, but won't work on all systems.
delay = 50
vga = normal  # force sane state
ramdisk = 0   # paranoia setting
# End LILO global section
# append the second ethernet card NE2000
append = " ether=5,0x360,eth1 "
# Linux bootable partition config begins
image = /vmlinuz
root = /dev/hda2
label = linux
# Linux bootable partition config ends
#
# DOS bootable partition config begins
other = /dev/hda1
label = dos
table = /dev/hda
# DOS bootable partition config ends

```

### B.2.2 /etc/rc.d/rc.inet1:

```

#! /bin/sh
#
# rc.inet1 This shell script boots up the base INET system.
#
# Version: @(#) /etc/rc.d/rc.inet1 1.01 05/27/93
#

```

```

HOSTNAME='cat /etc/HOSTNAME'

# Attach the loopback device.
/sbin/ifconfig lo 127.0.0.1
/sbin/route add -net 127.0.0.0

# IF YOU HAVE AN ETHERNET CONNECTION, use these lines below to configure the
# eth0 interface. If you're only using loopback or SLIP, don't include the
# rest of the lines in this file.

# Edit for your setup.
#IPADDR="150.131.250.111" # REPLACE with YOUR IP address!
#NETMASK="255.255.255.0" # REPLACE with YOUR netmask!
#NETWORK="150.131.251.0" # REPLACE with YOUR network address!
#BROADCAST="150.131.252.255" # REPLACE with YOUR broadcast address, if you
# have one. If not, leave blank and edit below.
#GATEWAY="150.131.251.10" # REPLACE with YOUR gateway address!

# Uncomment ONLY ONE of the three lines below. If one doesn't work, try again.
# /sbin/ifconfig eth0 ${IPADDR} netmask ${NETMASK} broadcast ${BROADCAST}
# /sbin/ifconfig eth0 ${IPADDR} broadcast ${BROADCAST} netmask ${NETMASK}
# /sbin/ifconfig eth0 ${IPADDR} netmask ${NETMASK}

# Uncomment these to set up your IP routing table.
#/sbin/route add -net ${NETWORK} netmask ${NETMASK}
#/sbin/route add default gw ${GATEWAY} metric 1

```

```

#/sbin/route add 150.131.252.0 gw ${GATEWAY} netmask ${NETMASK} metric 1

/sbin/ifconfig eth0 150.131.250.111 netmask 255.255.255.0 broadcast 150.131.250.255
/sbin/ifconfig eth1 150.131.252.111 netmask 255.255.255.0 broadcast 150.131.252.255
/sbin/route add 150.131.250.0 gw 150.131.250.111
/sbin/route add 150.131.251.0 gw 150.131.252.10
/sbin/route add 150.131.252.0 gw 150.131.252.111
/sbin/route add 150.131.251.0 gw 150.131.252.10

# End of rc.inet1

```

### B.3 The customized files on the 4.3BSD:

#### B.3.1 /etc/netstart:

```

#!/bin/sh -
#
# @(#)netstart 5.9 (Berkeley) 3/30/91
#
# These flags specify whether or not to run the daemons,
# and the flags to run them with
routedflags=-q
timedflags=NO
rwhod=NO
rstatd=NO

# NFSD_OPTS over-rides the default args for nfsd

```

```
# NFSD_OPTS='-u 255.255.255.0,192.124.139.0,4 -t 255.255.255.0,192.124.139.0'
```

```
# Change the following line to reflect your hostname
```

```
hostname=bsdunix
```

```
hostname $hostname
```

```
# NOTE: If you are using an /etc/resolv.conf file then you may need to  
# use internet numbers instead of host names in the following commands  
# (e.g., use 192.27.45.1 instead of myhost.domain).
```

```
# Uncomment lines corresponding to the interfaces you have installed,  
# you may also need to set a netmask if you are on a subnet.  
# Some adapters enable the BNC connector if given the link0 flag.
```

```
# If you have multiple interfaces replace $hostname with the name or number
```

```
#ifconfig we0 inet $hostname link0 # WD8003/8013 & 3COM 3C503 w/BNC
```

```
#ifconfig we0 inet $hostname # WD8003/8013 & 3COM 3C503
```

```
#ifconfig eo0 inet $hostname # 3COM 3C501
```

```
#ifconfig ep0 inet $hostname # 3COM 3C505 EtherLink Plus
```

```
#ifconfig el0 inet $hostname # 3COM 3C507
```

```
#ifconfig ef0 inet $hostname # 3COM 3C509 EtherLink III
```

```
#ifconfig ex0 inet $hostname # Intel EtherExpress 16
```

```
#ifconfig ne0 inet $hostname # Novell NE1000/NE2000
```

```
#ifconfig pe0 inet $hostname # Xircom Pocket Ethernet 2
```

```
#ifconfig rn0 inet $hostname __remotehost__ # RISCOm/N1 HDLC
```

```
# Sample hardwired SL/IP connection
```

```

# link0 means compress TCP traffic
# link1 means suppress ICMP traffic
# link2 means auto-enable TCP compression
#ifconfig sl0 $hostname __remotehost__ link2 up
#stty -f /dev/tty00 clocal
#slattach /dev/tty00 9600

# set the address for the loopback interface
ifconfig lo0 inet localhost

# use loopback, not the wire
#route add $hostname localhost

# configure a default route
#route add default __gateway__

#set up interface card and route table
ifconfig we0 150.131.250.110 netmask 255.255.255.0 broadcast 150.131.250.255
route add -net 150.131.250.0 150.131.250.110
route add -net 150.131.252.0 150.131.250.111
route add -net 150.131.251.0 150.131.250.111
#end of set up

```

### B.3.2 /etc/sendmail.cf:

```

# Copyright (c) 1983 Eric P. Allman
# Copyright (c) 1988 The Regents of the University of California.

```

```
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
# notice, this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
# notice, this list of conditions and the following disclaimer in the
# documentation and/or other materials provided with the distribution.
# 3. All advertising materials mentioning features or use of this software
# must display the following acknowledgement:
# This product includes software developed by the University of
# California, Berkeley and its contributors.
# 4. Neither the name of the University nor the names of its contributors
# may be used to endorse or promote products derived from this software
# without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS 'AS IS' AND
# ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
# FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
# OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
```

```
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
# SUCH DAMAGE.
```

```
#
```

```
# @(#)sendmail.cf 5.2 (Berkeley) 4/23/91
```

```
#
```

```
#
```

```
#####
```

```
#####
```

```
####
```

```
#### SENDMAIL CONFIGURATION FILE
```

```
####
```

```
#####
```

```
#####
```

```
#####
```

```
# local info #
```

```
#####
```

```
# file containing our internet aliases
```

```
Fw/etc/sendmail.cw
```

```
# uucp hostnames
```

```
# (local uucp host name goes here)
```

```
DU
```

```
# (local uucp host name aliases go here)
```

```
CU
```

```
# local UUCP connections
# (machines this machine contacts via uucp go here)
CV

#####
###  Setup Information  ###
#####

#####
#  General Macros  #
#####

# local domain name
# your domain here!
DDnet3

# Internet relay host -- machines in our domain that are not
# registered with the NIC will be "hidden" behind this relay machine
# with the % kludge, although SMTP delivery will still be performed
# by the sending machine.  Someday this will go away.
DAbsdunix

# UUCP relay host
DRdarkstar

# csnet relay host
```



DC

# bitnet relay host

DB

# my official hostname

Dj\$w

#####

# Classes #

#####

# Internal ("fake") domains that we use in rewriting

CIUUCP BITNET CSNET

#####

# Version Number #

#####

DZ1.37

#####

# Special macros #

#####

# my name

DnMAILER-DAEMON

```

# UNIX header format

DlFrom $g $d

# delimiter (operator) characters

Do.:%@!~/[]

# format of a total name

Dq$g$?x ($x)$

# SMTP login message

De$j Sendmail $v/$Z ready at $b

#####

# Options #

#####

# location of alias file

OA/etc/aliases

# wait up to ten minutes for alias file rebuild

Oa10

# substitution for space (blank) characters

OB.

# (don't) connect to "expensive" mailers

#Oc

# default delivery mode (deliver in background)

Odbackground

# temporary file mode

OF0600

# default GID

Og1

```

```
# location of help file
OH/usr/share/misc/sendmail.hf

# log level
OL9

# default network name
ONARPA

# default messages to old style
Oo

# queue directory
OQ/var/spool/mqueue

# read timeout -- violates protocols
Or2h

# status file
OS/var/log/sendmail.st

# queue up everything before starting transmission
Os

# default timeout interval
OT3d

# time zone names (V6 only)
OtPST,PDT

# default UID
Ou1

# wizard's password
OW*

# load average at which we just queue messages
Ox8

# load average at which we refuse connections
```

OX12

```
#####  
#   Message precedences   #  
#####
```

```
Pfirst-class=0  
Pspecial-delivery=100  
Pbulk=-60  
Pjunk=-100
```

```
#####  
#   Trusted users   #  
#####
```

```
Troot  
Tdaemon  
Tuucp
```

```
#####  
#   Format of headers   #  
#####
```

```
H?P?Return-Path: <$g>  
HReceived: $?sfrom $s $.by $j ($v/$Z)  
id $i; $b  
H?D?Resent-Date: $a
```

H?D?Date: \$a  
H?F?Resent-From: \$q  
H?F?From: \$q  
H?x?Full-Name: \$x  
HSubject:  
# HPosted-Date: \$a  
# H?l?Received-Date: \$b  
H?M?Resent-Message-Id: <\$t.\$i@\$j>  
H?M?Message-Id: <\$t.\$i@\$j>

#####  
### Rewriting Rules ###  
#####

#####  
# Sender Field Pre-rewriting #  
#####  
S1  
#R\$\*<\$\*>\$\* \$1\$2\$3 defocus

#####  
# Recipient Field Pre-rewriting #  
#####  
S2  
#R\$\*<\$\*>\$\* \$1\$2\$3 defocus

#####

```
# Final Output Post-rewriting #
```

```
#####
```

```
S4
```

```
R@ $@ handle <> error addr
```

```
# resolve numeric addresses to name if possible
```

```
R$*<@[$+]>$* $:$1<@[$2]>$3 lookup numeric internet addr
```

```
# externalize local domain info
```

```
R$*<$+>$* $1$2$3 defocus
```

```
R@$+:@$+:$+ @$1,@$2:$3 <route-addr> canonical
```

```
# UUCP must always be presented in old form
```

```
R$+@$-.UUCP $2!$1 u@h.UUCP => h!u
```

```
# delete duplicate local names
```

```
R$+%$=w@$=w $1@$w u%/host@host => u@host
```

```
R$+%$=w@$=w.$D $1@$w u%/host@host => u@host
```

```
#####
```

```
# Name Canonicalization #
```

```
#####
```

```
S3
```

```
# handle "from:<>" special case
```

```
R$*<>$* $@@ turn into magic token
```

```

# basic textual canonicalization -- note RFC733 heuristic here
R$*<$*<$*<$+>$*>$*>$* $4 3-level <> nesting
R$*<$*<$+>$*>$* $3 2-level <> nesting
R$*<$+>$* $2 basic RFC821/822 parsing

# make sure <@a,@b,@c:user@d> syntax is easy to parse -- undone later
R@$$,$+ @$1:$2 change all ", " to ":"

# localize and dispose of route-based addresses
R@$$:$$ @$>6<@$1>:$2 handle <route-addr>

# more miscellaneous cleanup
R$+ $:$>8$1 host dependent cleanup
R$+:$*;$+ @$1:$2;$3 list syntax
R$+:$*;$ @$1:$2; list syntax
R$+@$+ $:$1<@$2> focus on domain
R$+<$+@$+> $1$2<@$3> move gaze right
R$+<@$+> @$>6$1<@$2> already canonical

# convert old-style addresses to a domain-based address
R$+~$+ $1!$2 convert ~ to !
R$-!$+ @$>6$2<@$1.UUCP> resolve uucp names
R$+.$-!$+ @$>6$3<@$1.$2> domain uucps
R$+!$+ @$>6$2<@$1.UUCP> uucp subdomains
R$+%$+ $:$>9$1%$2 user%/host
R$+<@$+> @$>6$1<@$2> already canonical

```

R\$-. \$+ @\$>6\$2<@\$1> host.user

#####  
# special local conversions #  
#####

S6

R\$\*<@\$=w>\$\* \$:\$1<@\$w>\$3 get into u@\$w form  
R\$\*<@\$=w.\$D>\$\* \$:\$1<@\$w>\$3  
R\$\*<@\$=U.UUCP>\$\* \$:\$1<@\$w>\$3

#####  
# Change rightmost % to @. #  
#####

S9

R\$\*%\$\* \$1@\$2 First make them all @'s.  
R\$\*@\$\*@\$\* \$1%\$2@\$3 Undo all but the last.  
R\$\*@\$\* @\$1<@\$2> Put back the brackets.

#####  
### Mailers ###  
#####

#####  
#####  
####



#### Local and Program Mailer specification

####

#####

#####

Mlocal, P=/usr/libexec/mail.local, F=lsDFMmn, S=10, R=20, A=mail -r \$g -d \$u

Mprog, P=/bin/sh, F=lsDFMe, S=10, R=20, A=sh -c \$u

S10

R@ \$n errors to mailer-daemon

#####

#####

####

#### Local Domain SMTP Mailer specification

####

#### Messages processed by this specification are assumed to remain

#### the local domain. Hence, they can refer to hosts that are

#### not registered in the NIC host table.

####

#####

#####

Mtcpld, P=[IPC], F=mDFMueXLC, S=17, R=27, A=IPC \$h, E=\r\n

S17

```

# cleanup forwarding a bit
R$*<$*>$* $1$2$3 defocus

R$* $:$>3$1 canonicalize

R$*%$*<@$w> $:$>9$1%$2 user%localhost@localdomain

# pass <route-addr>'s through
R<@$+>$* @$<@$[$1$]>$2 resolve <route-addr>

# map colons to dots everywhere
R$*:$* $1.$2 map colons to dots

# output local host as user@host.domain
R$- @$1<@$w> user w/o host

R$+<@$w> @$1<@$w> this host

R$+<@$=w> @$1<@$w> or an alias

R$+<@$-> $:$1<@$[$2$]> ask nameserver

R$+<@$w> @$1<@$w> this host

R$+<@$-> @$1<@$2.$D> if nameserver fails

# if not local, and not a "fake" domain, ask the nameserver
R$+<@$+.$~I> @$1<@$[$2.$3$]> user@host.domain

R$+<@[$+]> @$1<@[$2]> already ok

# output fake domains as user%fake@relay

R$+<@$+.BITNET> @$1%$2.BITNET<@$B> user@host.bitnet

R$+<@$+.CSNET> @$1%$2.CSNET<@$C> user@host.CSNET

```

```
R$+<@$+.UUCP> @$2!$1<@$w> user@host.UUCP
```

S27

```
# cleanup
```

```
R$*<*$>*$ $1$2$3 defocus
```

```
R$* $:$>3$1 now canonical form
```

```
R$%*$<@$w> $:$>9$1%$2 user%localhost@localdomain
```

```
# pass <route-addr>'s through
```

```
R<@$+>*$ @$<@$[$1$]>$2 resolve <route-addr>
```

```
# map colons to dots everywhere
```

```
R$*:$* $1.$2 map colons to dots
```

```
# output local host as user@host.domain
```

```
R$- @$1<@$w> user w/o host
```

```
R$+<@$w> @$1<@$w> this host
```

```
R$+<@$=w> @$1<@$w> or an alias
```

```
R$+<@$-> $:$1<@$[$2$]> ask nameserver
```

```
R$+<@$w> @$1<@$w> this host
```

```
R$+<@$-> @$1<@$2.$D> if nameserver fails
```

```
# if not local, and not a "fake" domain, ask the nameserver
```

```
R$+<@$+.$~I> @$1<@$[$2.$3$]> user@host.domain
```

```
R$+<@[$+]> @$1<@[$2]> already ok
```

```
# output fake domains as user%fake@relay
```

```
R$+<@$+.BITNET> $$1%$2.BITNET<@$B> user@host.BITNET
```

```
R$+<@$+.CSNET> $$1%$2.CSNET<@$C> user@host.CSNET
```

```
R$+<@$+.UUCP> $$2!$1 user@host.UUCP
```

```
#####
```

```
#####
```

```
####
```

```
#### Internet SMTP Mailer specification
```

```
####
```

```
#### Messages processed by this specification are assumed to leave
```

```
#### the local domain -- hence, they must be canonical according to
```

```
#### RFC822 etc. This means that machines not registered with
```

```
#### the NIC must be hidden behind our Internet relay.
```

```
####
```

```
#####
```

```
#####
```

```
Mtcp, P=[IPC], F=mDFMueXLC, S=14, R=24, A=IPC $h, E=\r\n
```

```
S14
```

```
# pass <route-addr>'s through
```

```
R<@$+>$* $$<@$[$1$]>$2 resolve <route-addr>
```

```
# map colons to dots everywhere
```

R\$\*:\$\* \$1.\$2 map colons to dots

# output local host in user@host.domain syntax

R\$- \$1<@\$w> user w/o host

R\$+<@\$=w> \$:\$1<@\$w> this host

R\$+<@\$-> \$:\$1<@\$[\$2\$]> canonicalize into dom

R\$+<@\$-> \$:\$1<@\$2.\$D> if nameserver fails

R\$+<@\$=N.\$D> @\$1<@\$2.\$D> nic-reg hosts are ok

R\$+<@\$\*.\$D> @\$1%\$2.\$D<@\$A> else -> u%/h@gateway

# if not local, and not a "fake" domain, ask the nameserver

R\$+<@\$+.\$~I> @\$1<@\$[\$2.\$3\$]> user@host.domain

R\$+<@[\$+]> @\$1<@[\$2]> already ok

# output internal ("fake") domains as "user%/host@relay"

R\$+<@\$+.BITNET> @\$1%\$2.BITNET<@\$B> user@host.BITNET

R\$+<@\$+.CSNET> @\$1%\$2.CSNET<@\$C> user@host.CSNET

R\$+<@\$+.UUCP> @\$2!\$1<@\$w> user@host.UUCP

S24

# put in <> kludge

R\$\*<\*\$>\*\$ \$1\$2\$3 defocus

R\$\* \$:\$>3\$1 now canonical form

# pass <route-addr>'s through

```

R<@$+>*$ @$<@$[$1$]>$2 resolve <route-addr>

# map colons to dots everywhere.....
R$*:$* $1.$2 map colons to dots

# output local host in user@host.domain syntax
R$- $1<@$w> user w/o host
R$+<@$=w> $:$1<@$w> this host
R$+<@$-> $:$1<@$[$2$]> canonicalize into dom
R$+<@$-> $:$1<@$2.$D> if nameserver fails
R$+<@$=N.$D> @$1<@$2.$D> nic-reg hosts are ok
R$+<@$*.$D> @$1%$2.$D<@$A> else -> u%/h@gateway

# if not local, and not a "fake" domain, ask the nameserver
R$+<@$+.$~I> @$1<@$[$2.$3$]> user@host.domain
R$+<@[$+]> @$1<@[$2]> already ok

# Hide fake domains behind relays

R$+<@$+.BITNET> @$1%$2.BITNET<@$B> user@host.BITNET
R$+<@$+.CSNET> @$1%$2.CSNET<@$C> user@host.CSNET
R$+<@$+.UUCP> @$2!$1 user@host.UUCP

#####
#####

####

#### UUCP Mailer specification

```

#####

#####

#####

Muucp, P=/usr/bin/uux, F=DFMhuU, S=13, R=23, M=100000,

A=uux - -r -z -a\$f -gC \$h!rmail (\$u)

S13

R\$+ \$:\$>5\$1 convert to old style

R\$\*<@\$=w>\$\* \$1<@\$w>\$2 resolve abbreviations

R\$\*<@\$->\$\* \$1<@\$2.\$D>\$3 resolve abbreviations

R\$+<@\$+> \$2!\$1 uucpize (no @'s in addr)

R\$w!\$+ \$1 strip local name

R\$+ \$:\$U!\$1 stick on our host name

R\$=U!\$-!\$- \$:\$1!\$2@\$3.\$D ucbvax!user@host.domain

S23

R\$+ \$:\$>5\$1 convert to old style

R\$\*<@\$=w>\$\* \$1<@\$w>\$2 resolve abbreviations

R\$\*<@\$->\$\* \$1<@\$2.\$D>\$3 resolve abbreviations

R\$+<@\$w> \$U!\$1 a!b@here -> here!a!b

R\$=U!\$+ \$2 here!a!b -> a!b

# sanity ... should not happen.

R\$=U.\$D!\$+ \$2 strip local name.domain

#####

#####

```

####
#### Provide Backward Compatibility
####
#####
#####

#####
# General code to convert back to old style names #
#####
S5

R$+<@w> $1 strip host
R$+<@$.UUCP> $2!$1 u@host.UUCP => host!u

#####
### Rule Zero ###
#####

#####
#####

####
#### RULESET ZERO PREAMBLE
####
#### The beginning of ruleset zero is constant through all
#### configurations.
####
#####

```



#####

S0

# first make canonical

R\$\*<\$\*>\$\* \$1\$2\$3 defocus

R\$+ \$:\$>3\$1 make canonical

# handle special cases

R\$\*<@[+]>\$\* \$:\$1<@[[\$2]]>\$3 numeric internet addr

R\$\*<@[+]>\$\* \$#tcp\$@[2]\$: \$1@[2]\$3 numeric internet spec

R\$+ \$:\$>6\$1

R\$-<@\$w> \$#local\$: \$1

R@ \$#error\$:Invalid address handle <> form

# canonicalize using the nameserver if not internal domain

R\$\*<@\$\*.\$~I>\$\* \$:\$1<@\$[\$2.\$3]>\$4

R\$\*<@\$->\$\* \$:\$1<@\$[\$2]>\$3

R\$\*<@\$->\$\* \$:\$1<@\$2.\$D>\$3 if nameserver fails

# now delete the local info

R<@\$w>:\$\* @\$>0\$1 @here:... -> ...

R\$\*<@\$w> @\$>0\$1 ...@here -> ...

#####

# End of ruleset zero preamble #

#####

```

#####
###  Machine dependent part of Rule Zero  ###
#####

# resolve local UUCP connections
R<@=V.UUCP>:$+ $#uucp$@1$:$2 @host.UUCP:...
R$+<@=V.UUCP> $#uucp$@2$:$1 user@host.UUCP

# resolve fake top level domains by forwarding to other hosts
R$*<@$.BITNET>$* $#tcp$@B$:$1<@2.BITNET>$3 user@host.BITNET
R$*<@$.CSNET>$* $#tcp$@C$:$1<@2.CSNET>$3 user@host.CSNET

# forward non-local UUCP traffic to our UUCP relay
R$*<@$.UUCP>$* $#tcp$d$@R$:$1<@2.UUCP> uucp mail

# resolve SMTP traffic
R$*<@$.D>$* $#tcp$d$@2.D$:$1<@2.D>$3 user@host.ourdomain
R$*<@+>$* $#tcp$@2$:$1<@2>$3 user@host.ourdomain

# remaining names must be local
R$+ $#local$:$1 everything else

```

## B.4 The customized files on the SunOS:

### B.4.1 /etc/rc.local:

```

#
# @(#)rc.local 1.112 90/09/14 SMI; from UCB 4.3
#
PATH=/bin:/usr/bin:/usr/etc:/usr/ucb; export PATH
domainname `cat /etc/defaultdomain`

#
# hostname now set in rc.boot
#
echo -n 'starting rpc and net services:'
if [ -f /usr/etc/portmap ]; then
portmap; echo -n ' portmap'
fi
if [ -f /usr/etc/ypserv -a -d /var/yp/`domainname` ]; then
ypserv; echo -n ' ypserv'
#
# Master NIS server runs the XFR daemon
#
# ypxfrd; echo -n ' ypxfrd'
fi
if [ -d /var/yp ]; then
if [ -f /etc/security/passwd.adjunct ]; then
ypbind -s; echo -n ' ypbind'
else
ypbind; echo -n ' ypbind'
fi
fi

```

```

if [ -f /usr/etc/keyserv ]; then
keyserv; echo -n ' keyserv'
fi

if [ -f /usr/etc/rpc.yppupdated -a -d /var/yp/'domainname' ]; then
rpc.yppupdated; echo -n ' yppupdated'
fi

# set the netmask from NIS if running, or /etc/netmasks for all ether interfaces

ifconfig -a netmask + broadcast + > /dev/null

#
# If we are a diskless client, synchronize time-of-day with the server.
# Else, if applicable, run the router daemon. Note that for better
# performance, we don't enable the router daemon for diskless clients.
#
# At the same time, terminate the currently printing line (for prettiness).
#
server='grep ":[.*[ ][ ]*/[ ]" /etc/fstab | sed -e "/^#/d" -e "s/:[.*//"'
if [ "$server" ]; then
echo '.'
intr -a rdate $server
else
#if [ -f /usr/etc/in.routed ]; then
# in.routed; echo -n ' routed'
#fi
echo '.'

```

```

fi

#
# The following will mount /tmp if set up in /etc/fstab.  If you want to use
# the anonymous memory based file system, have an fstab entry of the form:
# swap /tmp tmp rw 0 0
# Make sure that option TMPFS is configured in the kernel
# (consult the System and Network Administration Manual).
#
# mount /tmp

intr -a mount -vat nfs

echo -n 'starting additional services:'
if [ -f /usr/etc/in.named -a -f /etc/named.boot ]; then
in.named; echo -n ' named'
fi
if [ -f /usr/etc/biod ]; then
biod 4; echo -n ' biod'
fi
echo '.'

```

```

# syslogd doesn't belong here, but needs to be started before the others.
# It needs to be started after NIS, though, so it can find the "syslog"
# udp service.
if [ -f /usr/etc/syslogd ]; then
echo 'starting system logger'
rm -f /dev/log
syslogd
fi

#
# Default is to not do a savecore
#
#mkdir -p /var/crash/'hostname'
# echo -n 'checking for crash dump... '
#intr savecore /var/crash/'hostname'
# echo ''

if [ -f /dev/sky ]; then
skycrc /usr/lib/sky.unicode
fi

if [ -f /dev/fpa ]; then
/usr/etc/fpa/fpa_download -d -r
fi

if [ -f /dev/fpa ]; then
/usr/etc/fpa/fparel
fi

```

```

if [ -f /usr/etc/gpconfig ]; then
/usr/etc/gpconfig -f -b
fi
if [ -f /dev/dialbox ]; then
dbconfig /dev/dialbox
fi
# add the gateway 150.131.252.10
/usr/etc/route add 0 150.131.252.10 1
route add net 150.131.250.0 150.131.252.111 1
echo -n 'starting local daemons:'
if [ -f /usr/etc/auditd ]; then
auditd; echo -n ' auditd'
fi
if [ -f /usr/lib/sendmail -a -f /etc/sendmail.cf ]; then
(cd /var/spool/mqueue; rm -f nf* lf*)
/usr/lib/sendmail -bd -q1h; echo -n ' sendmail'
fi
if [ -d /tftpboot -a -f /tftpboot/sun2.bb ]; then
ndbootd; echo -n ' ndbootd'
fi
#
# if /etc/exports file exists become nfs server
#
if [ -f /etc/exports ]; then
> /etc/xtab
exportfs -a
nfsd 8 & echo -n ' nfsd'

```

```

if [ -f /etc/security/passwd.adjunct ]; then
# Warning! Turning on port checking may deny access to
# older versions (pre-3.0) of NFS clients.

rpc.mountd
echo "nfs_portmon/W1" | adb -w /vmunix /dev/kmem >/dev/null 2>&1
else
rpc.mountd -n
fi
fi
#
# if /tftpboot exists become a boot server
#
if [ -d /tftpboot ]; then
echo -n ' rarpd'; \
rarpd -a
rpc.bootparamd
fi
#
# start up status monitor and locking daemon if present
#
if [ -f /usr/etc/rpc.statd ]; then
rpc.statd & echo -n ' statd'
fi
if [ -f /usr/etc/rpc.lockd ]; then
rpc.lockd & echo -n ' lockd'
fi
#

```



```

# start up authentication daemon if present and if adjunct file exists
#
if [ -f /usr/etc/rpc.pwdauthd -a -f /etc/security/passwd.adjunct ]; then
rpc.pwdauthd & echo -n ' pwdauthd'
fi
#
# start up the automounter
#
if [ -f /usr/etc/automount ]; then
automount && echo -n ' automount'
fi
echo '.'
#
# Build the link-editor fast directory cache.
#
if [ -f /usr/etc/ldconfig ]; then
ldconfig; echo "link-editor directory cache"
fi

```

## B.5 The customized file /etc/hosts on each machine:

```

#
# hosts This file describes a number of hostname-to-address
# mappings for the TCP/IP subsystem. It is mostly
# used at boot time, when no name servers are running.
# On small systems, this file can be used instead of a
# "named" name server. Just add the names, addresses

```

```
# and any aliases to this file...
#
# Version: @(#) /etc/hosts 2.00 04/30/93
#
# Author: Fred N. van Kempen, <waltje@uwalt.nl.mugnet.org>
#

# For loopbacking.
127.0.0.1 localhost
150.131.250.110 bsdunix.net3 bsd
150.131.250.111 linux.net3 linux
150.131.252.111 darkstar.net2 darkstar
150.131.252.112 fusion.net2 fusion
150.131.252.10 NetWare.net2 netware
50.131.251.10 pcware.net1 pcware
# End of hosts.
```

## Bibliography:

1. Douglas E. Comer.[1991]. *Internetworking with TCP/IP Volume I & II*. Prentice-Hall, Inc..
2. Barry Nance.[1990]. *Network Programming in C*. Que Corporation.
3. Thomas W. Madron.[1994]. *Local Area Network*. Joh Wiley & Sons, Inc..
4. Carl Malamud.[1992]. *Analyzing Novell Networks*. Van Nostrand Reinhold.
5. Charles G. Rose.[1990]. *Programmer's Guide to NetWare*. McGraw-Hill, Inc..
6. Michael Santifaller.[1994]. *TCP/IP and ONC/NFS internetworking in a UNIX environment*. Addison-Wesley(Deutschland) GmbH.
7. Evi Nemeth.[1995]. *UNIX System Administration*. Prentice Hall PTR
8. *NetWare documentation*.
9. *SunOS documentation*.
10. *Linux documentation*.